Algoritmos y Estructuras de Datos
Guía de Trabajos Prácticos Nro. 2

[Las guías deberán ser presentadas en un archivo .cpp con todas las funciones pedidas. Sólo se aceptaran éstas a través de la plataforma. En cada algoritmo desarrollado debe especificarse el orden del mismo.]

1. Escribir una función ***int maxtree(tree<int> &T)*** que retorna el máximo de las etiquetas de un árbol.

2. Escribir una función ***int suma(tree<int> &T)*** que calcula la suma de las etiquetas de un árbol.

3. Escribir una función ***int altura(tree<int> &T)*** que retorne la altura de un árbol.

4. Escribir una función ***int cuantospares(tree<int> &T)*** que retorna la cantidad de nodos con etiquetas pares que tiene un árbol.

5. Escribir una función ***int prof_par(tree<int> &T)*** que retorna la profundidad máxima de un nodo con etiqueta par.

6. Escribir una función ***int cuentacaminos(tree<int> &T, int l)*** que, dado un árbol ordenado orientado A, cuenta cuantos caminos de longitud ***l*** tiene el árbol.

7. Escribir una función ***void prevorden(tree<int> &T, list<int> &L)*** que, retorna en ***L,*** la lista de los nodos en orden previo.

8. Escribir una función ***void postorden(tree<int> &T, list<int> &L)*** que, retorna en ***L,*** la lista de los nodos en orden posterior.

9. Escribir una función ***void path_of_largest(tree<int> &AT,list<int> &L)***; que, dado un árbol ordenado orientado A, retorna en L el camino que se obtiene recorriendo el árbol hacia abajo, siempre por el hijo más grande. Por ejemplo, si A=(3 (4 6 (7 8 9)) (5 2 4 6)) entonces path_of_largest debe retornar L=(3 5 6). Si para un nodo el valor más grande de los hijos está repetido, entonces el camino puede seguir por cualquiera de ellos. Por ejemplo si A=(3 (4 5 6) 4 2) entonces path_of_largest() puede retornar L=(3 4 6) o L=(3 4).

10. Se tienen las listas oprev(n) y opost(n) que dan las posiciones en los órdenes previo y posterior, respectivamente, de cada nodo n de un árbol. Describa un algoritmo que diga si un nodo i es un antecesor o no de un nodo j, para cualquier par de nodos i, j.

11. Escribir una función ***void cumsum(btree<int> &T)***; (por suma acumulada) que dado un árbol binario (AB) T modifica el valor de los nodos interiores, de manera que resulta ser la suma de los valores de sus hijos más el valor que había en el nodo antes de llamar a cumsum(). Los valores de las hojas no son modificados, y los valores de los nodos interiores resultan ser la suma de todos los valores del sub árbol del nodo antes de llamar a cumsum. Ejemplo: Si T=(1 5 (2 3 7 (11 4 2))) entonces después de llamar cumsum(T) debe quedar T=(35 5 (29 3 7 (17 4 2))).

12. Dado un árbol binario, escribir una función ***bool es_completo(btree<int> &T)***; retorna true si el árbol es completo, es decir, todos sus nodos interiores tienen los dos hijos.

13. Dado un árbol binario, escribir una función ***double completitud(tbree<int> &T)***; retorna la relación entre el número de nodos interiores completo y el número total de nodos interiores. Para un árbol completo debe retornar 1.

14. Escribir una función ***bool is_full(btree<int> &T)*** que retorna si una árbol es o no “lleno”. Recordemos que un árbol es lleno, si todos sus niveles están completos. Se puede definir en forma recursiva de la siguiente manera:

- El árbol vacío es lleno.
- El árbol no vacío es lleno, si sus dos hijos son llenos y tiene la misma altura.

Sugerencia: Escribir una función auxiliar ***bool is_full(btree<int> &T,btree<int>::iterator n, int &height)***; que retorna por height la altura del árbol.

15. Dado un AB T encontrar, la profundidad máxima de los nodos que satisfacen un cierto predicado (profundidad condicionada). Por ejemplo, si T=(6 (7 9 (3 1)) 2), entonces depth_if(T,even) debe retornar 1, ya que el nodo par a mayor profundidad es 2, mientras que depth_if(T,odd) debe retornar 3, ya que el nodo impar a máxima profundidad es 1.
Consigna: Escribir una función int depth_if(btree<int> &T,bool (*pred)(int)); que realiza la tarea indicada.
Ayuda: La función recursiva auxiliar debe retornar la máxima profundidad de un nodo que satisface el predicado o -1 si el arbol está vacío o ningún nodo satisface el predicado. Entonces, dadas las profundidades condicionadas dl, dr de los hijos la profundidad se puede expresar en forma recursiva como

$$depth(n) = \begin{cases} -1 \text{ ; si n es } \lambda \\ max(dr, dl) + 1 \text{ ; si } max(dr, dl) \geq 0 \\ 0 \text{ ; si n satisface el predicado} \\ -1 \text{ ; si n no satisface el predicado} \end{cases}$$